THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE ARTEMIS JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 269335 AND FROM SPECIFIC NATIONAL PROGRAMS AND/OR FUNDING AUTHORITIES.

MBAT

ARTEMIS Project MBAT:
Combined Model-based Analysis and Testing of Embedded Systems

Specification of Model-Based Analysis Methods and Tools

D_WP2.4_2

Deliverable	Specification of model-based analysis methods and tools		
Confidentiality	Public	**Deliverable type**	Report
Project	MBAT	**Contract Number**	269335
Deliverable File Name	D_WP2.4_2	**Date**	2012-11-16
Status	final, submitted to ARTEMIS-JU	**Version**	1.0
Contact Person	Brian Nielsen	**Organisation**	AAU
Phone	+45 9940 8883	**E-Mail**	bnielsen@cs.aau.dk

Authors

Name	Company	E-Mail
Brian Nielsen	AAU	bnielsen@cs.aau.dk
Gianna Bellè	AAU	gianna@cs.aau.dk
Markus Pister	ABS	pister@absint.com
Orlando Ferrante	ALES	orlando.ferrante@ales.eu.com
Marco Carloni	ALES	marco.carloni@ales.eu.com
Dejan Nickovic	AIT	dejan.nickovic@ait.ac.at
Helene Le Guen	A4T	helene.leguen@all4tec.net
Stefan Häusler	BTC	stefan.haeusler@btc-es.de
Olivier Bouissou	CEA LIST	olivier.bouissou@cea.fr
Mehrdad Saadatmand	ENEA	mehrdad.saadatmand@xdin.com
Claudia Schett	IFAT	schett.external@infineon.com
Raluca Marinescu	MDU	raluca.marinescu@mdh.se
Cristina Seceleanu	MDU	cristina.seceleanu@mdh.se
Sven Sieverding	OFFIS	sven.sieverding@offis.de
Vesal Vojdani	TUM	vojdanig@in.tum.de
Christian Schwarzl	VIF	christian.schwarzl@v2c2.at

Reviewers (partners who have reviewed this deliverable)

Name	Company	E-Mail
Bernhard K. Aichernig	TU Graz (TUG)	aichernig@ist.tugraz.at
Ralf Bogusch	Cassidian	Ralf.Bogusch@cassidian.com
Anjelika Votintseva	Siemens	anjelika.votintseva@siemens.com

Distribution (to whom this deliverable has been distributed)

Name / Role	Company	Level of confidentiality	Type of deliverable
MBAT partners		PU	R
Antinio Vecchio, Project Officer	ARTEMIS-JU	PU	R

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.1	27-09-2012	First version (template) and call for contributions	*
0.2	17-10-2012	Collect all the contributions.	*
0.3	08-11-2012	Merged reviewed versions and first update according to the comments.	*
0.4	16-11-2012	Update according to review	*
1.0	16-11-2012	Reviewed	*
	19-11-2012	Finalized & submitted to ARTEMIS-JU	*

CONTENT

Figures

Tables

1 Introduction

1.1 Overview, Purpose and Scope

This deliverable provides descriptions of tool features to be developed and/or evaluated during the development of the MBAT RTP. Furthermore it specifies the functionalities of the tool versions available at major project milestones M2, M3, resp. M4.

Month/Milestone	Content	Validated by
10 / M1 / RTPv0	Methods and Tools for the Fast Track Demonstrator (RTP V0) have been provided.	SP3 through successful integration into the Fast Track RTP
18 / M2 / RTPv1	First versions of MBA and MBT tools are available demonstrating the capabilities of these tools.	SP3 through successful integration into the Fast Track RTP
30 / M3 / RTPv2	Second versions of MBA and MBT tools are available offering the full functionality and fully integrated in the MBAT environment.	SP3 through successful integration into the Fast Track RTP
36 / M4	Optimized versions of MBA and MBT tools are available taking into account the latest feedback from the end users.	Industrial End Users. After projects during exploitation activities

Table 1-1: Milestones according to the Technical Annex

The table above shows that most of the functionalities should be implemented for milestones M2 and M3. The purpose of M4 is polishing functionalities and doing updates that are not mandatory for the final evaluation. The plans here have been made on basis of MBAT goals and initial analysis of use cases. The plans may be revised as experiences are collected from application in use cases and development of the RTP. Consequently, in the following, empty entries in the partner's individual plan overview tables mean that no (new) features are planned as per the date of this deliverable), not implying that they will not be improved or possibly added.

1.2 Relation to Other Deliverables

- *D_WP2.4_3_1 Prototype Tools for Model-based Analysis*
  The tool components and the reports describing them are provided in another series of deliverables [WP2.4b, 2012] due in months 10, 18, 30, 36 respectively.

  Specifically, the deliverable [WP2.4b, 2012] has described each tool component in terms of tool category, purpose, unique features, provider, functional description, input/outputs, tool control, how to obtain it, and references for further information. Also, the tool components were classified there according to their supported level of abstraction and emphasis on safety, functionality, or quantitative properties. Hence, the focus of the present deliverable

is to concentrate on the development plans, and the features expected to become available via MBAT.

- *D_WP2.3_2 Specification of Model-based Test Case Generation and Execution Methods and Tools*
- *D_WP2.3_3_1 Prototype tools for Model-based Test Case Generation and Execution*

As MBAT also deals with model-based testing and how this can be combined with analysis it is also relevant to consult the "sister" deliverables [WP2.3a] and [WP2.3b].

- *D_WP1.4_1 Automotive Use cases & demonstrators specification*
- *D_WP1.5_1Aerospace Use cases & demonstrators specification*
- *D_WP1.6_1Rail Use cases & demonstrators specification*
- *D_WP2.1_1_1 Refined Requirements*
- *D_WP1.3_1_1 Measurement plans*

The analysis tool components and features are to a large extent driven by the needs of the MBAT use cases, and evaluated in context of these. They also consider other industrial requirements and needs as identified by [WP2.1a]. Remark that because MBAT is use case driven, the exact set of implemented features may differ from the ones planned here.

- *D_WP2.4_1_1 Requirements for use of WP2.4 results in the RTP*
- *D_WP3.2_2_1 Specifications for RTP Interoperability*

Finally, the analysis tools are not going to be used in isolation but in conjunction with other MBAT tools and with other (software) engineering tools, and must interoperate with these in the RTP.

2 Plans for Analysis Tool Components

2.1 Aalborg University - AAU

UPPAAL (see http://www.uppaal.org/) is a tool suite for modelling, simulating, verifying and testing real-time systems modelled as network of timed automata communicating via channel synchronization and extended with discrete variables.

The following table shows the tool components based on UPPAAL framework that are provided by AAU.

	M2	M3	M4
UPPAAL model checker and UPPAAL-SMC	SMC for hybrid systems Simulink plug-in Analysis result export Service enabling and OSLC/RTP integration v1	Model management and traceability UPPAAL and SMC integration SMC for EAST-ADL Support for UML notation Parameter analysis Service enabling and OSLC/RTP integration v2	Optimized features (TBD)
ECDAR	Available for experimental use		
opaal+LTSmin		Experimental combination of model checking and static analysis	

Table 2-1: AAU tool functionalities

2.1.1 UPPAAL model checker and UPPAAL-SMC

We treat these tool components together as they are closely related and will be fully integrated in a (near) future release.

2.1.1.1 Tool description

The main feature of the UPPAAL model checker is its ability to model-check *timed* systems with the help of efficient symbolic techniques. The specification language is a subset of TCTL, an intuitive branching logic that allows the user to check for reachability, safety, and liveness properties. In particular, the tool can check for absence of deadlocks. The user can also query for infimum or supremum of expressions, which can be useful for computing, e.g., worst-case response times for schedulability analysis. When checking a property, the user gets as an output the result (satisfied or not) and may get a witness trace or counter example, e.g., specifying a shortest or fastest one.

UPPAAL-SMC extends timed automata to stochastic and hybrid automata by supporting different rates on clocks, extended arithmetic on clocks, weighted transitions, and rates on locations to give distributions on how the automata should delay. The user can now ask more quantitative properties to the model-checker to study the performance of models. The tool generates random runs and, based on statistical methods, and gives answers with guaranteed levels of confidence. The tool offers advanced visualizations of probability distributions, accumulated probabilities, and can plot different values of expressions.

2.1.1.2 Development plans within MBAT

In addition to general tool improvements, the following features targeting MBAT goals and use case needs (AAU is focused on the use cases AUT0_UC4_DAI and RAIL_UC3_SIE) are planned to be developed:

- **UPPAAL and UPPAAL-SMC integration:** Currently, these two tool components are separated as seen from both users and developers. To ease use and maintenance these will be integrated in one solution. A main challenge from the industrial end-use perspective is to develop a common simulator. The built-in simulator/animator in the UPPAAL GUI currently displays symbolic states and traces using clock constraints. While technically most accurate, this method has proven difficult to use for many engineers, and hence a more user friendly concrete state simulator is foreseen. In contrast, the engine for SMC is based on concrete execution runs, but there is currently no graphical simulator/animator for SMC. However, but due to its probabilistic and hybrid extensions it is not trivial to promote this directly to the GUI. Hence, the task involves significant development work.
- **Model management:** Working with models and model checking often involves making several model modifications and variants (with different parameter values, model elements, constants, etc.) to determine how to model a feature in the best way or satisfy a desired property. For complex tasks and models as required by the use cases this quickly results in a zoo of model files and produced artefacts and status of (dis-) satisfied properties, whose relationship is hard to trace. We plan to develop support for selection and maintenance of model variants and versions, traceability of verification results with models, tool configuration, and properties. Both internal support and external support (for the RTP traceability) is foreseen.
- **Support for UML notation:** We will define a subset of MARTE UML state machines that can be imported into UPPAAL as timed automata templates, that preserves a meaningful UML semantics (in the context of formal analysis of timing properties) and that can be efficiently verified using the UPPAAL model-checker. In addition we plan to define model annotations for hybrid and stochastic behaviour to be used with UPPAAL-SMC. We plan to implement import and export of UPPAAL templates to XMI. This tool feature will enable UPPAAL to be used by use case engineers familiar with the UML notation, and enables formal model check of (restricted) UML state machines models.
- **SMC for hybrid systems**: Development of support in UPPAAL-SMC for hybrid and weighted timed automata (significant progress have already been made on this in the first year of MBAT). These will further be extended with features for the refinement check between (high-level) timed models and (low-level) hybrid automata. We will enable specification of hybrid behaviour using differential equations.
- **UPPAAL Simulink plug-in:** We plan to develop a feature that enables (high level) timed automata models to be simulated within Matlab/Simulink (via an S-function). This is for instance useful for validating by simulation a high level design in context of a (possibly

existing and/or rich) dynamic model made in Matlab/Simulink, thus providing a link between formal analysis and testing or simulation.

- **Parameter analysis:** The behaviour of models and consequently any analysis result and utility depends on model input parameters representing variability in system values, constants, or system configurations etc. Quite often it is insufficient to analyse a single "instantiated" model with concrete values of the parameters. One goal is to identify the parameters for which the system is correct or performs best. Another goal is to determine how sensitive or robust the behaviour of a system is to changes in a particular parameter configuration. We will define syntax and semantics for parameterized timed automata models, and develop support for their analysis using parameter sweep techniques.
- **SMC for EAST-ADL:** Support for performance analysis of (timed) EAST-ADL models via the translation offered by ViTAL and UPPAAL PORT, as requested by the AUTO_UC1_VOL use case.
- **Analysis result export:** To further strengthen the links to external (test and analysis) tools and the RTP we plan to develop exports of model-checking results like satisfied properties, counter examples, and SMC results/graphs in well-specified (XML-based) format(s).
- **Service enabling:** Enabling of the UPPAAL engine (and editor/simulator GUI) as a (web) service, enabling integration in other environments, specifically enabling OSLC/RTP integration. A similar task is defined for our model-based testing engine, together creating an environment for model-based analysis and testing in the same modelling framework.

2.1.2 ECDAR

2.1.2.1 Tool Description

ECDAR (Environment for Compositional Design and Analysis of Real-time systems) is a tool based on the UPPAAL framework that is designed for compositional verification of models based on step-wise refinements (see http://people.cs.aau.dk/adavid/ecdar/). Here the user defines different models of specifications that can be combined together with a set of operators such as parallel composition or conjunction. Then these specifications can be checked for 1) consistency, i.e., if there exist some implementations that meet these specifications; or 2) for refinement. Different refinements can be checked step-wise on small models to conclude a property on a larger model without checking the larger model directly (this could be too expensive). This extension uses the same interface as UPPAAL-TIGA.

2.1.2.2 Development plans within MBAT

This component will be offered for experimental application to use cases requiring specification of timed interfaces and compositional reasoning. Developments will be based on the specific requirements of the use case.

2.1.3 opaal+LTSmin

2.1.3.1 Tool Description

"opaal+LTSmin" (see http://opaal-modelchecker.com/) is a prototype model checker designed to enable trying out new model checking features or concepts for timed automata model checking. It is a combination of two model-checking tools, opaal and LTSmin. opaal is a prototyping framework for timed automata model checking. LTSmin supports various manipulations of labelled transition systems as well as model checking. LTSmin works as a backend for several existing verification tools. LTSmin supports multi-core explicit model checking and reachability. Using the opaal model

checker support for timed automata, LTSmin have been extended to support multi-core reachability of timed automata models.

2.1.3.2 Development plans within MBAT

To support multi-core model checking of liveness properties needed by safety critical systems we plan on adding support for LTL model checking. To allow better scalability we would like to extend opaal + LTSmin with support for a joining operator to enable model checking that can take advantage of abstractions as known from *static program analysis* using abstract interpretation. Furthermore, we would like to enable model checking of even larger models by implementing support for distributed model checking of timed automata in LTSmin.

2.2 AbsInt Angewandte Informatik - ABS

AbsInt provides advanced development tools for embedded systems, and tools for validation, verification and certification of safety-critical software. For this purpose, safety guarantees for non-functional properties of models (e.g. Simulink or SCADE models), such as absence of possible runtime errors, memory consumption and worst-case execution time have to be determined. The AbsInt tools support the determination of such properties by three abstract interpretation-based tools:

- the static analyzer **Astrée**
- **aiT** WCET Analyzer
- **StackAnalyzer**

The following table lists features/methods/functions that are planned to be implemented by the AbsInt analyzers. Partly, the features are already available in the current version 12.10.

	M2	**M3**	**M4**
Astrée	Runtime error detection, Global assertions, Alarm density report, Scenario builder, Analysis wrapper generator, BTC-ET tool coupling, TargetLink coupling, OSLC compliance	Analysis history support, Per function density, Program slicing	On-demand product enhancements, AIS export, SCADE coupling
aiT	WCET bound determination, BTC-ET coupling, TargetLink coupling, OSLC compliance	On-demand product enhancements	On-demand product enhancements
StackAnalyzer	Worst-case stack consumption computation, BTC-ET coupling TargetLink coupling, OSLC compliance	On-demand product enhancements	On-demand product enhancements

Table 2-2: ABS tool functionalities

For Astrée, there are already quite a lot of planned enhancements, which will be described in the next sections alongside the assignment to specific use case scenarios (and thus use cases). Where Astrée is used in several use cases, for the other tools, aiT and StackAnalyzer, no concrete enhancements have been specified yet; their taking part in the Ricardo use case is currently being evaluated. Based on the results of this evaluation, specific product enhancements might be scheduled throughout the project.

2.2.1 The Static Analyzer Astrée

2.2.1.1 Tool description

Astrée is a static program analyser that proves the absence of run-time errors (RTE) in safety-critical embedded applications written or automatically generated in C.

It analyses structured C programs with complex memory usages according to the C99 standard. It targets embedded applications as found in earth transportation, nuclear energy, medical instrumentation, aeronautics and space flight, in particular synchronous control/command such as electric flight control.

Astrée reports any definite and also potentials:

- division by zero,
- out-of-bounds array indexing,
- erroneous pointer manipulation and dereferencing (NULL, uninitialized and dangling pointers),
- integer and floating-point arithmetic overflow,
- violation of optional user-defined assertions to prove additional run-time properties (similar to assert diagnostics known from the C programming language specification),
- code fragments it can prove to be unreachable under any circumstances (either definitely unreachable code or unreachable due to a detected runtime error without any feasible continuation),
- read access to uninitialized variables.

The analyser is sound for floating-point computations and handles them precisely and safely. It takes all possible rounding errors into account.

Moreover, it offers powerful annotation mechanisms, which enable the user to make external knowledge available to Astrée, or to selectively influence the analysis precision for individual loops or data structures. Detailed messages and an intuitive GUI help the user understand alarms about potential errors. Then, true runtime errors can be fixed, or, in case of a false alarm, the analyser can be tuned to avoid them. These mechanisms allow performing analyses with very few or even zero false alarms.

2.2.1.2 Development plans within MBAT

The features listed in Table 2-2 are shortly introduced in the following descriptions.

Runtime-error detection

The main feature of Astrée is the detection of potential runtime errors in the analysed code as described above. Therefore, this feature is listed here for the sake of completeness and addresses the following scenarios:

- WP1.4_DAI_UC01_SC07
- WP1.4_DAI_UC01_SC08

Global assertions

The directive `__ASTREE_global_assert((V , [l, h]));`, where V is a global variable of integer or float type and [l, h] is an interval, tells the analyzer that the value of the variable V should be between [l, h] during the whole program execution. In order to ensure this, the analyzer checks at every assignment to the variable V whether the variable is still within its bounds. If the variable

leaves the specified range the analyzer outputs an alarm and reduces the range of V to the asserted bounds.

Global assertions address scenario WP1.4_DAI_UC02_SC01.

Alarm density report

The alarm density window shows a visualization of the number of alarms per function relative to the number of alarms in other functions. To this end, the window region is divided into a set of rectangles, each of which represents a function with at least one alarm. The size of a rectangle depends on the number of alarms in this function relative to the number of alarms in other functions. That is, functions with larger rectangles have more alarms than functions with smaller rectangles. Consequently, the representation allows to easily identifying hotspots in the analysis where most alarms have been reported.

Each rectangle is labelled with the name of its function and its number of alarms. Functions with only few alarms may be represented by very small rectangles without label. However, the corresponding information is still accessible via the tooltip, which are displayed when positioning the mouse pointer on the rectangle. Finally, one can directly jump to the source code of a function by clicking on its corresponding rectangle.

Additionally to the graphical representation of this feature, the alarms are written to a report file so that the information is available within the A&T model and other tools can read it. A future version of this feature will be able to show density information per function of the analysed code.

The alarm density feature addresses scenario WP1.4_DAI_UC02_SC02.

Scenario builder

A scenario in the context of Astrée is an analysis project that is derived from an original analysis project by reduction. This process of reduction is only defined for analysis projects that use an infinite main loop for modelling a synchronous program execution. A reduced project then consists of

- the option settings from the original project,
- an __astree_main__ procedure with an infinite main loop,
- code for maintaining invariants on selected variables,
- the code of the selected functions, including all sub-functions and necessary type definitions,

and

- code for calling the selected functions.

The goal of creating a reduced scenario is to simplify the investigation of alarms in synchronous programs. For example, if an alarm is raised in a certain function of the synchronous software under analysis, one may define a scenario by selecting only this function and all variables that are inputs to this function. The generated scenario then repeatedly calls the selected function while maintaining the invariant on the inputs that has been determined by the full analysis. If the same alarm is raised in the reduced scenario, it can be investigated in isolation, allowing for faster experimentation because of the reduced analysis time.

The scenario builder opens in a separate window, which displays

- a list of variables with information about value ranges, types and scope of declaration, in the top-left corner,

- a list of functions in the bottom-left corner, and
- a list of scenario items on the right-hand side.

These elements only exist if

- the analysed code contains an infinite loop,
- the option "Dump (infinite) loop invariants including unrolled iterations" has been enabled, and
- the analysis has finished successfully, in particular, the infinite main loop has been fully analysed.

If all requirements are met, a set of variables and functions can be selected from the lists on the left-hand side of the window and added to the scenario items list on the right-hand side using the two Add buttons (located above the lists). Scenario items are identified as functions or variables using a V or F prefix character. Pushing the Build scenario button generates a new analysis project based on the scenario definition. The generated project can then be immediately opened by clicking on the "Yes" button in the following dialog box.

The list of variables comprises only those variables that are visible in the main loop.

The extraction of reduced scenarios is fully automatic if the selected functions do not take any parameters and if the variables have simple scalar, plain struct, or array types. Otherwise function parameters are omitted and must be added manually by the user. Similarly, information about invariants for pointer types is generated only in comments. The corresponding code for maintaining the invariants must be written by the user. The same strategy applies for cases where the invariant produced by Astrée is too complex for automatic processing.

Analysis wrapper generator

Astrée supports separate analysis of software components. As these components are often represented by dedicated C functions that are called within a reactive loop. Then, assumptions about input variable ranges may have to be available to the analyser, which can be achieved via specific directives. Furthermore, assumptions about output ranges can be statically checked by assert directives. Sometimes, this reactive loop is not coded explicitly but realized by the operating system via system callbacks, etc. Then, a wrapper function has to be written in order to ease the analysis of the system.

The analysis wrapper generator supports this task by taking a list of ranges for global input variables, a list of initialization functions, and a list of tasks. From this information it generates C code that reflects the execution model of the analysed code. The generated code can be used as the entry point for the analysis.

The following scenarios are addressed by the analysis wrapper generation feature:

- WP1.4_DAI_UC02_SC04
- WP1.4_DAI_UC01_SC03

BTC-ET tool coupling

In the context of MBAT, the static analyser Astrée and the dynamic testing tool BTC-ET (cf. [WP2.4a, 2012]) are planned to be coupled at different levels of integration. The first solution, which is already available, here is a *process integration level combination*. The test tool BTC-ET automatically creates test models for automatic test generation for arbitrary levels of the system design modelled in TargetLink. The combination can leverage from this feature and gain an automatic analysis model construction for automated application of the three AbsInt tools. It hence

gives an essential workflow optimization for applying the analysis tools. This makes working with analysis tools much more convenient and flexible, and makes analysis iterations much faster, thus saving time and costs.

BTC-ET has a complete view of the model-based design (TargetLink) of the system under test. Additionally, it knows the relations and has traceability between the model components and system's C code as is has been generated by TargetLink. BTC-ET is able to automatically generate a set of test vectors that fulfil structural coverage criteria and execute them against all levels (MiL, SiL, PiL). Test results are then compared to each other to detect derivation between model and code. In order to apply Astrée on the generated C code various preparation steps have to be performed before actually running a concrete analysis. First an analysis model has to be prepared, some settings for Astrée have to be specified, and the analysis run has to be launched. These steps are fully automated by BTC-ET by providing dedicated analysis wrappers. By leveraging from these wrappers the concrete analysis can be run with a single mouse click. After analysis completion the generated analyses reports are visualized within a unified reporting framework showing in a single view both testing results and analysis conclusions.

Through the MBAT project, a more methodological integration will be achieved, where the BTC-ET tool is able to reconstruct test cases that trigger alarms reported by Astrée.

This coupling directly addresses the following scenarios:

- WP1.4_DAI_UC01_SC02
- WP1.4_DAI_UC01_SC06

TargetLink and SCADE coupling

One of the overall goals for the Astrée development in MBAT is to take plant/test model information into account for the analysis. Environmental information, e.g. on value ranges of global variables, should be extracted from analysis models. Moreover, the analysis precision and error reporting facilities should be improved generally. One concrete realization is the extraction of any suitable information from TargetLink data dictionaries or SCADE models. For the first, an implementation already exists, i.e. Astrée is able to read in data dictionaries generated by TargetLink and generate appropriate annotations for the analyser. To achieve a similar support for SCADE models, discussions with Esterel Technologies are ongoing.

Analysis history support

Astrée analyses are planned to support analysis history management functionality. In detail, Astrée can handle multiple revisions of an analysis project. This allows to compare current analysis results with results of former configurations and also to revert projects to earlier states. By default, every project has at least one revision. If a project has more than one revision, the list of revisions can be displayed by unfolding the project entry in the Open dialog. New revisions can be created in the Open dialog by right-clicking on an entry in the list of available projects and selecting `Create new revision` from the context menu. By creating a new revision, the current state of the selected revision is copied. The selected revision then becomes a so-called base revision, i.e., a revision with child revisions. Base revisions can no longer be modified but it is possible to derive further revisions from them in order to revert the analysis to an earlier state. By deriving several revisions from the same base revision, the analysis history is no longer linear. The Analysis History dialog can be used in order to navigate the resulting tree structure. The dialog can be opened by right-clicking on a revision of the project and selecting the Analysis History entry from the context menu. It graphically displays the revision tree and allows to

- open leaf revisions,
- create new revisions,

- delete single revisions, and
- delete complete sub-trees.

To open a revision, select a node via left-click and then push the dedicated Open button. All other operations are accessible via a context menu which is opened by right-clicking on a node in the revision graph.

A future version of this history management will be able to highlight differences between analysis revisions.

The analysis history feature addresses scenario WP1.4_DAI_UC01_SC08.

Program Slicing

There is no program slicer in Astrée, yet. In a first version, program slices shall be computed without taking pointer aliasing into account. Further, it shall be possible to extract the source code of program slices. In later versions, pointer aliasing shall be considered.

The program slicing feature addresses the following scenarios:

- WP1.4_DAI_UC01_SC02
- WP1.4_DAI_UC01_SC07

Automata domain

In addition to the already existing analysis domains within the Astrée analyzer, a new domain is planned to be developed in co-operation with ENS. The new domain shall enable fewer false alarms when analyzing programs that involve automata.

The automata domain feature addresses the following scenarios:

- WP1.4_DAI_UC01_SC04
- WP1.4_DAI_UC01_SC07

AIS export

Where Astrée is analysing the software at the C-code level, aiT and StackAnalyzer examine compiled and fully linked binary executables for the determination of worst-case bound on the runtime and stack consumption. Partly, information on loop iteration counts and value ranges of variables can be determined using Astrée but cannot be statically derived from the executable. Astrée then might communicate certain information to an aiT/StackAnalyzer analysis by generating suitable AIS annotations. AIS is the AbsInt specification language used to externally incorporate information for the particular WCET/Stack analysis.

The AIS export feature addresses scenario WP1.4_DAI_UC01_SC03.

Interoperability/OSLC compliance

The above described tool coupling with BTC-ET is currently realized as point-to-point connections, where BTC-ET generates an analysis wrapper and needed setup information for Astrée. Until milestone M2, this direct connection is planned to be replaced by corresponding OSLC adapters.

Covered core requirements

The following core requirements are addressed either by already existing functionality of Astrée or any of the above described developments:

- RQ-SP2-WP2.1-1
- RQ-SP2-WP2.1-7
- RQ-SP2-WP2.1-28
- RQ-SP2-WP2.1-33
- RQ-SP2-WP2.1-71
- RQ-SP2-WP2.1-75
- RQ-SP2-WP2.1-9
- RQ-SP2-WP2.1-20
- RQ-SP2-WP2.1-30
- RQ-SP2-WP2.1-70

2.2.2 The WCET Analyzer aiT

2.2.2.1 Tool description

aiT WCET Analyzers (one tool for each supported hardware architecture, see below) statically compute tight bounds for the worst-case execution time (WCET) of tasks in real-time systems. They directly analyze binary executables and take the intrinsic cache and pipeline behaviour into account. For this, the control-flow graph of the input binary is reconstructed by decoding into an intermediate representation, called CRL (Control-flow Representation Language). Based on this description, the analyser performs abstract interpretation of program execution based on a micro-architectural execution model of the underlying target architecture which computes execution times at the basic block or instruction level. The results of the micro-architectural analyses are combined in a so-called generic path analysis which computes the worst-case program path. This is the execution path of the analysed program that leads to the computed worst-case execution time bound.

Each aiT analyzer is developed specifically for the underlying hardware architecture because of the above mentioned architectural execution model. Currently, AbsInt supports the following architectures: AM486, Infineon C16x/ST10, Texas Instruments C33, Atmel ERC32, Motorola HC11 and HCS12, Intel I386DX, LEON2, LEON3, Motorola PowerPCs 5xx, 55xx, 56xx, 603e, 7448, 750, 755, Infineon TriCore and NEC V850. The HCS12 architecture is used by the Ricardo use case (UC A5).

2.2.2.2 Development plans within MBAT

BTC-ET tool coupling

Analogously to Astrée (see above), aiT will be coupled with the testing tool BTC-ET, i.e., BTC-ET generates a suitable project setup for aiT and reads the analysis results after having executed aiT. For the benefits of such a tool coupling, please have a look at the corresponding description above in Section 2.2.1.2.

The application of aiT in general and the tool coupling with BTC-ET especially addresses the following two scenarios:

- WP1.4_RIC_UC01_SC02
- WP1.4_RIC_UC01_SC04

TargetLink and SCADE coupling

One of the overall goals for the aiT development in MBAT is to take plant/test model information into account for the analysis. Environmental information, e.g. on value ranges of global variables,

should be extracted from analysis models. Moreover, the analysis precision and error reporting facilities should be improved generally. One concrete realization is the extraction of any suitable information from TargetLink data dictionaries or SCADE models. For the first, an implementation already exists, i.e. aiT is able to read in data dictionaries generated by TargetLink and generate appropriate annotations for the analyser. To achieve a similar support for SCADE models, discussions with Esterel Technologies are ongoing.

Interoperability/OSLC compliance

The above described tool coupling with BTC-ET is currently realized as point-to-point connections, where BTC-ET generates an analysis wrapper and needed setup information for aiT. Until milestone M2, this direct connection is planned to be replaced by corresponding OSLC adapters.

Covered core requirements

The following core requirements are addressed either by already existing functionality of aiT or any of the above described developments:

- RQ-SP2-WP2.1-1
- RQ-SP2-WP2.1-7
- RQ-SP2-WP2.1-28
- RQ-SP2-WP2.1-33
- RQ-SP2-WP2.1-71
- RQ-SP2-WP2.1-75
- RQ-SP2-WP2.1-9
- RQ-SP2-WP2.1-20
- RQ-SP2-WP2.1-30
- RQ-SP2-WP2.1-70

2.2.3 StackAnalyzer

2.2.3.1 Tool description

StackAnalyzer automatically determines the worst-case stack usage of the tasks in an application. Its features are:

- Detailed and precise information on user-stack and system-stack usage.
- Stack analysis for all hierarchy levels: routines, basic blocks, assembly instructions.
- Control-flow reconstruction directly from binary code. This means that StackAnalyzer will not be confused by potential flaws in the debug information.
- Seamless integration with aiT for worst-case execution time analysis in a single intuitive user interface called a³.

The control-flow graph of the input binary is reconstructed by decoding into an intermediate representation, called CRL (Control-flow Representation Language). Based on this description, the analyser performs abstract interpretation of program execution based on a micro-architectural execution model of the underlying target architecture.

Each StackAnalyzer is developed specifically for the instruction set semantics of the underlying hardware architecture because of the above mentioned architectural execution model. Currently, AbsInt supports the following architectures: AM486, Fujitsu ARC, ARM, Infineon C16x/ST10, Texas Instruments C33, Atmel ERC32, Renesas H8, Motorola HC11, Motorola HCS12, Intel I386DX, LEON2, LEON3, Motorola M68k, Motorola PowerPC, Infineon TriCore, NEC V850 and Intel X86 (flat memory model and real mode). This list is very similar to the one given for aiT (see

above). Although both tools share parts of their tool chain, there does not exist a StackAnalyzer for each architecture supported by aiT and vice versa.

2.2.3.2 Development plans within MBAT

BTC-ET tool coupling

Analogously to Astrée (see above), StackAnalyzer will be coupled with the testing tool BTC-ET, i.e., BTC-ET generates a suitable project setup for StackAnalyzer and reads the analysis results after having executed stack analysis. For the benefits of such a tool coupling, please have a look at the corresponding description above in Section 2.2.1.2.

The application of StackAnalyzer in general and the tool coupling with BTC-ET especially addresses the following two scenarios:

- WP1.4_RIC_UC01_SC02
- WP1.4_RIC_UC01_SC04

TargetLink and SCADE coupling

Due to a partly shared tool chain, the planned TargetLink and SCADE coupling is the same as it is described above for aiT.

Interoperability/OSLC compliance

The above described tool coupling with BTC-ET is currently realized as point-to-point connections, where BTC-ET generates an analysis wrapper and needed setup information for Astrée. Until milestone M2, this direct connection is planned to be replaced by corresponding OSLC adapters.

Covered core requirements

The following core requirements are addressed either by already existing functionality of StackAnalyzer or any of the above described developments:

- RQ-SP2-WP2.1-1
- RQ-SP2-WP2.1-7
- RQ-SP2-WP2.1-28
- RQ-SP2-WP2.1-33
- RQ-SP2-WP2.1-71
- RQ-SP2-WP2.1-75
- RQ-SP2-WP2.1-9
- RQ-SP2-WP2.1-20
- RQ-SP2-WP2.1-30
- RQ-SP2-WP2.1-70

2.3 Advanced Laboratory on Embedded Systems - ALES

Advanced Laboratory on Embedded Systems (ALES) develops methods and tools for the validation and verification of complex embedded systems. ALES contributes to the WP with the enhancement of the FormalSpecs Verifier (FSV) framework covering both the formal specification of requirements using the Block-based Contract Language (BCL) and the formal verification of contract satisfaction relation as summarized in Table 2-3.

	M2	M3	M4
FormalSpecs Verifier for Contract Verification	BCL language enhancements and support for use-case driven extensions Formal verification of contract satisfaction relation (v1) RTP (v1) Integration	Contracts analysis for test case reduction (v1) Formal verification of contract satisfaction relation (v2) RTP (v2) Integration	Contracts analysis for test case reduction (v2)

Table 2-3: ALES tool functionalities

2.3.1 FormalSpecs Verifier – Contract Verification

The FormalSpecs Verifier (FSV) is a model based framework for the verification of complex embedded systems providing a MATLAB Simulink based environment for the description and verification of requirements modeled as contracts. A contract for a system, or a component of the system, under design is a formal representation of a requirement made of two parts: the *promises*, which are properties that must be guaranteed, and the *assumptions*, which define under what conditions the promises must hold. . Contracts have been exploited by the SPEEDS European project[1] and further developed in the SPRINT European project[2]. The Contract Based Design paradigm defines several relations among contracts and between an implementation and its contracts. In the context of MBAT ALES will focus on the satisfaction relation. When the implemented component behaves correctly with respect to a contract specification we say that the component satisfies that contract. Checking satisfaction, therefore, amounts to making sure that a component will provide the stated promises when used in a context that does not violate the assumptions. Figure 2-1 represents the verification flow supported by the tool. As a first step the designer provides a functional description of the system under analysis using the MATLAB Simulink environment. Then he/she specifies the requirements as contracts formalizing the assumptions and the promises using a user friendly language provided with the FSV framework. The FSV tool automatically elaborates the model formally verifying the satisfaction relation between the system under analysis and the specified contracts. In case the satisfaction analysis is violated, a harness model is produced showing the violation of the contract.

[1] http://www.speeds.eu.com
[2] http://www.sprint-iot.eu/

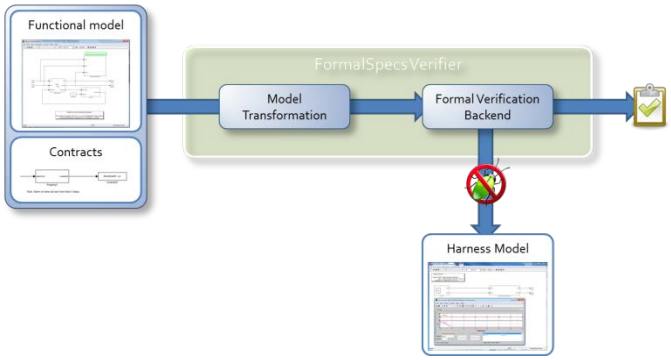

Figure 2-1: FormalSpecs Verifier verification flow

The FormalSpecs Verifier framework supports the formalization of requirements as contracts using a specific pattern based language called BCL (Block-based Contract Language). The BCL relies on the concept of assertion that represents a formalized sentence. Assertions can be composed using specific operators to build complex formal sentences and contracts are specified using this mechanism identifying for each of them the assumption and promise assertions. The BCL conceptually extends the Contract Specification Language (CSL) developed in the SPEEDS European project adopting its pattern-based nature in order to provide a user-friendly specification mechanism. Figure 2-2 represents an example of contract specification using BCL. Atomic patterns are used to assert conditions that must hold for the assumptions or promises or to compose them. Finally the contract block identifies the assumption and the promise. The FSV tool currently supports a limited subset of the BCL language mainly covering patterns expressing safety assertions.

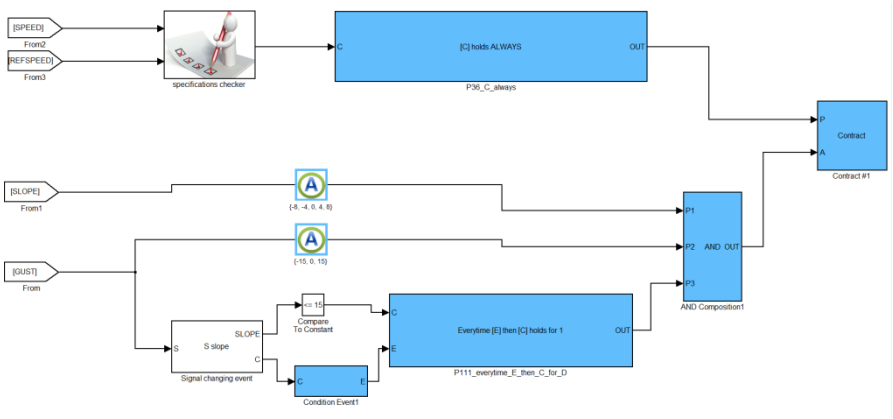

Figure 2-2: Example of BCL pattern specification in Simulink

2.3.1.1 Development plans

During the MBAT project the FSV tool will be enhanced in several directions as described in this paragraph.

BCL language extension

The FSV will be enhanced to fully support the BCL language including the formal verification *of liveness* assertions as well as their legal compositions. In addition, a subset of the language will be identified and extended in order to provide a unique formalism for the specification of contracts

suitable for both ATG and formal verification techniques. Finally, the BCL language will be extended to support the specific needs of the use case providers extending the language with specific elements when needed.

Formal verification of contract satisfaction relation

The FSV tool will be extended to fully support the formal verification of the satisfaction relation of a functional description of a system or a component and the related contracts. Contracts will be formalized using the BCL language.

The FSV tool will be enhanced to satisfy use-case providers' needs formulating the use-case specific analyses in the contract-based design framework and executing them using the FSV tool. As MBAT project we aim at supporting the Ansaldo STS (ASTS) and Alenia Aermacchi use cases among the others.

Support of ASTS use case analyses using the contract satisfaction relation

The ASTS use case is described in the deliverable [WP1.6a, 2012]. Specific analyses are formulated for the identification of design errors and reduction of testing effort. As tool provider we aim at supporting the analyses of ASTS using the contract satisfaction verification technique implemented in the FSV tool.
In particular, we are going to consider two of the required analyses identified in that deliverable to define the following activities:

- Formal verification of Reduction Rules: each reduction rule (RR) will be modelled as a contract and given as input, along with the functional model of the system, to the FSV tool. The FSV tool will formally check the satisfaction of the contract, hence the effectiveness of the reduction rule..
- Formal verification of safety properties. Specific system states are modelled as unreachable using the contract-based formalism and the FSV tool will be applied to verify the satisfaction of such safety contracts by the functional model of system.

Contracts analysis for test case reduction

To fully exploit the synergies between static analysis and testing we plan to use formal verification techniques to optimize the test case generation process by applying formal analysis on contracts in order to obtain formal proof that can be used to drive the ATG process. The goal of this process is to reduce the total number of generated tests improving the coverage of each case. This step will be addressed enhancing the interoperability between the FormalSpecs Verifier Property Verification and ATG engine.

RTP Integration

The FSV-ATG will be integrated in the MBAT RTP platform to enable full interoperability with the MBAT platform developing the needed adapters.

2.4 All4tec – A4T

All4tec is involved in MBAT project as a tool provider. The following tools participate in MBAT.

	M2	M3	M4
MaTeLo	Requirement pairs coverage	Analysis of test campaign results	Impact analysis Providing analysis results to the RTP
Safety Architect		Adding SIL level notion in safety analysis	Providing analysis results to the RTP
MaTeLo and other tools		Checking inconsistencies between models	Analysis of test cases generated on several levels

Table 2-4: A4T tool functionalities

2.4.1 MaTeLo

2.4.1.1 Tool description

MaTeLo is mainly a model-based test generator tool. Nevertheless MaTeLo performs some analysis, like analysis on the test model, analysis on the generated test cases and analysis on test results.

MaTeLo is an available commercial tool.

2.4.1.2 Development plans within MBAT

Requirement pairs coverage

Requirements are linked in MaTeLo to the test model elements (e.g. transition or input stimulation). A requirement can be linked to several elements. Associations between requirement and model elements have a property (necessary or sufficient) used to determine the coverage of requirement. For example, if a requirement is associated with two elements with the property "necessary", the requirement is covered by a test case if the two elements are included in the model path corresponding to the test case.

When test cases are generated, the list of the requirements covered by the test case is calculated.

This new functionality is used to determine before generation if each pair of requirements can be covered by a single test case. If the list is not empty perhaps there is some missing information in the test model.

Analysis on test campaign result

This analysis is done by taking into consideration the test model, the test cases and the test results. Some indicators are calculated:

- Test model and requirement coverage
- Relevance of testing strategy
- Reliability
- Error localization

These indicators will take into account product line and safety aspects.

Impact Analysis

After a modification of the requirements, MaTeLo produces a report containing information of the modifications to be done on the model, on the test cases and on the test execution.

2.4.2 MaTeLo and other tools

Some activities are not limited only to MaTeLo and include also other testing (and analysis) tools.

Checking inconsistencies between models

In order to be efficient in Test and Analysis, it will be necessary to make some analysis between different models, in order to see if they are compliant and correct.

This verification will be done between different models

- OVM (Orthogonal Variability Model)
- Test model (MaTeLo)
- Design model (Diversity, Simulink?)
- Code model (PathCrawler?)
- ...

The way to do it is not clearly decided yet. Perhaps the stimulation trace (on design model) will be used to detect inconsistencies, or an analysis made in parallel on models can determine if the models are coherent.

Analysis of test cases generated on several levels

The first interest of this task is to detect if some tests applied on different levels (System, Integration, Code) are redundant.

The main issue is to reduce test effort by analysing the tests executed on the other levels.

2.4.3 Safety Architect

2.4.3.1 Tool description

Safety Architect is a tool to make a safety analysis on a component diagram.

2.4.3.2 Development plans within MBAT

Providing analysis results to the other tools via the RTP

Adding an OSCL adaptor to Safety Architect.

Adding SIL level notion in safety analysis

Adding the quantitative analysis to the current propagation engine to complete the relevancy of the analysis.

2.5 Austrian Institute of Technology - AIT

AIT will provide the following tools.

	M2	M3	M4
DTF	Support for additional communication protocols Interoperability with automatic test case generator Implementation of new concepts for evaluation of simulation results	Automatic configuration based on descriptions of the system architecture and behaviour Derivation of requirements from possibly incomplete initial sets of requirements Integration into a workflow management system Graphical User Interface	--

Table 2-5: AIT tool functionalities

2.5.1 Data Time Flow Simulator (DTF)

2.5.1.1 Tool description

The Data Time Flow Simulator (DTF) is a discrete-event simulation environment for model based design, evaluation and performance analysis of electronic control systems. A prototype of the DTF was developed and validated in the course of the ARTEMIS project POLLUX, which is related to the development of embedded control systems for the next generation of electric cars. The DTF provides a set of primitive building blocks, called the *elements*. Typical elements are sensors, actuators, processors, communication controllers and others. Each element has a pre-defined functionality, which can be changed by incorporation of user defined code to adapt the element to user specific needs. Elements are grouped together to form more complex structures, like electronic control units (ECUs).

A typical electronic control system consists of a set of spatially separated ECUs, executing a distributed application, exchanging communication via a communication medium, e.g. a Controller Area Network (CAN) bus. For the performance analysis of such systems, the DTF focuses on the validation of requirements with regard to

- **Bus load**
  The peak load of the communication system.
- **Control signal latencies**
  The propagation time of control signals from sensors to actuators.

The aim of the DTF simulator is the identification of an architecture for the electronic control system that is as simple as possible and as complex as necessary and that fulfils all requirements. For this purpose, different variants of the system can be modelled and the advantages and disadvantages of the variants can be compared and weighted up against each other. The goal is to meet all relevant design decisions as early as possible in the development process, ideally before any hardware or software has been built. Such approach helps to avoid costly re-design activities

due to the detection of substantial design weaknesses in later development phases, like in the test and integration phase.

For the DTF which was developed in the course of the POLLUX project, the focus is on the modelling and simulation of automotive electronic control systems.

In the course of the MBAT project, the DTF simulator is deployed in Use Case 1 (BBW Brake-by-wire). For RTPv0, the following goals have been reached:

- Generation of a DTF simulation model of a hardware validator of the BBW system built by Volvo. The DTF model was transformed manually on the base of architectural and behavioural descriptions of the validator.
- Interoperability of the DTF with the test case generator MaTeLo from ALL4TEC.
- Assessment of the maximum bus load of this architecture using the generated test cases from the MaTeLo tool.

Based on these results, the development plans for additional DTF tool functionalities listed in Table 2-5 are described in detail.

2.5.1.2 Development plans within MBAT

Support for additional communication protocols

The current implementation of the BBW validator uses the CAN communication protocol which is already supported by the DTF. For future versions of the validator, the Ethernet protocol is intended to be used. Therefore, the DTF has to be extended to support this communication protocol.

Motivation: Driven by use case UC1

Interoperability with automatic test case generator

For RTPv0, the MaTeLo tool from ALL4TEC is used to generate test cases for DTF simulation experiments with the validator. The interface to the MaTeLo tool has to be refined such that the timing behaviour of the validator is considered.

MBAT goal addressed: Interoperability

Implementation of new concepts for evaluation of simulation results

This step aims at the development of efficient methods for data extraction and visualization. Since the DTF simulator shall be able to simulate full drive cycles, which have a duration of several thousands of seconds real-time (like e.g. the New European Driving Cycle for the assessment of the emission levels of car engines and fuel economy in passenger cars), large amounts of data are generated. This data has to be analysed in an efficient way to be able to extract data of interest in reasonable time. Furthermore, different sources of data have to be combined to be able to produce meaningful visualizations of the simulation results.

Motivations: Adaptation to industrial needs, Enhancement of tool practicability

Automatic configuration based on descriptions of the system architecture and behaviour

In the BBW use case, descriptions of the system architecture and the system behaviour are available in EAST-ADL format. EAST-ADL is an Architecture Description Language (ADL) for automotive embedded systems. Currently, the DTF configuration is created manually on the base of these system descriptions. However, one aim of the BBW use case is to create the DTF

configurations fully automatically. This requires the implementation of a converter from EAST-ADL into the DTF simulator configuration format.

MBAT goals addressed: Interoperability

Derivation of requirements from possibly incomplete initial sets of requirements

For the evaluation of the performance of a system architecture, the DTF simulator validates requirements with regard to bus load and control signal latencies. These requirements have to be known. However, in early development phases, not all requirements may be known in detail. Therefore, the DTF shall be able to compute derived requirements from incomplete sets of initial requirements. One example for a derived requirement is the maximum execution time for a control task in an electronic control unit, derived from the maximum control signal latency along an action chain from a sensor to an actuator, where the electronic control unit is part of this action chain.

MBAT goals addressed: Combination of test and analysis

This development goal is related to the following feature.

Integration into a workflow management system

Since the DTF simulator needs a set of requirements as an input, there has to be a mechanism to retrieve these requirements from a repository. In the BBW use case, a workflow manager (WEFACT) shall provide, among others, a requirement repository, derived from architectural and behavioural system descriptions. Therefore, the DTF has to be integrated into this workflow management system. For communication with the workflow manager, the open standard OSLC (Open Service for Lifecycle Collaboration) is intended to be used. OSLC is a set of open standards that aims at simplification of interoperability between tools from different vendors.

MBAT goals addressed: Interoperability, combination of test and analysis, OSLC compliance

Graphical User Interface

Currently, a typical DTF configuration is a human-readable file in a proprietary format. Depending on the system complexity, this configuration may become very large. Although a parser is provided that provides mechanisms for decomposition, like the possibility to define different system parts in different files that are included from a top-level configuration file, considerable amounts of typing work has to be done. These efforts shall be reduced by implementation of a graphical user interface that will allow creating systems hierarchically from a set of building blocks. Furthermore, the connections between different system parts shall be visualized to facilitate the validation of the correct “cabling” of system components.

Motivation: Improvement of usability

2.6 BTC Embedded Systems - BTC

	M2	M3	M4
BTC Embedded Specifier	-	Formal requirement specification for “Simulation based formal Verification”	--
BTC Embedded Tester	--	Simulation based formal Verification	-

Table 2-6: BTC tool functionalities

2.6.1 BTC EmbeddedSpecifier

2.6.1.1 Tool description

Key idea of BTC EmbeddedSpecifier is to provide out-of-the-box a set of pre-defined specification patterns that can easily be used to formally specify informal requirements. Patterns have a graphical nature and are thus much better to understand than many other formalisms, while formal semantics underlies the pattern; but hidden for the engineer. Additionally, they are declarative rather than operational. This means that engineers describe in a declarative way what the system should do, but not how. A pre-defined specification pattern library together with a tool supporting the engineer to develop the formal specification makes this process step much easier for a typical engineer.
Method and tool will be developed to guide a user during the complex conversion phases from an informal functional requirement specification, to a uniquely defined and correctly formalized requirement expressed as a graphical Observer Pattern. The method is used to incrementally, stepwise refine informal requirements to formal requirements and adapts itself to arbitrary development processes. C-Observers can be automatically generated out of Patterns. C-Observers are basically C-functions which return a boolean value for stating the validity of the related formal specification. The tool will interact with BTC EmbeddedTester that provides automatic test generation based on C-observers derived from formalized requirements.

2.6.1.2 Development plans

The tool will be enhanced to enable simulation-based formal verification both on model level (Simulink, TargetLink) and code level to verify formal specifications of requirements. Combined with EmbeddedTester, it will enable simulation based formal verification of requirements. BTC EmbeddedSpecifier will enable engineers to specify formal requirements both on function models (FM) like Simulink and implementation models (IM) like TargetLink, as described in Figure 2-3. C-Observer generation will be enhanced so that they can be generated also for models, not only for code (see Production Code Host and Target within Figure 2-3), enabling earlier formal verification of requirements than it is currently possible.

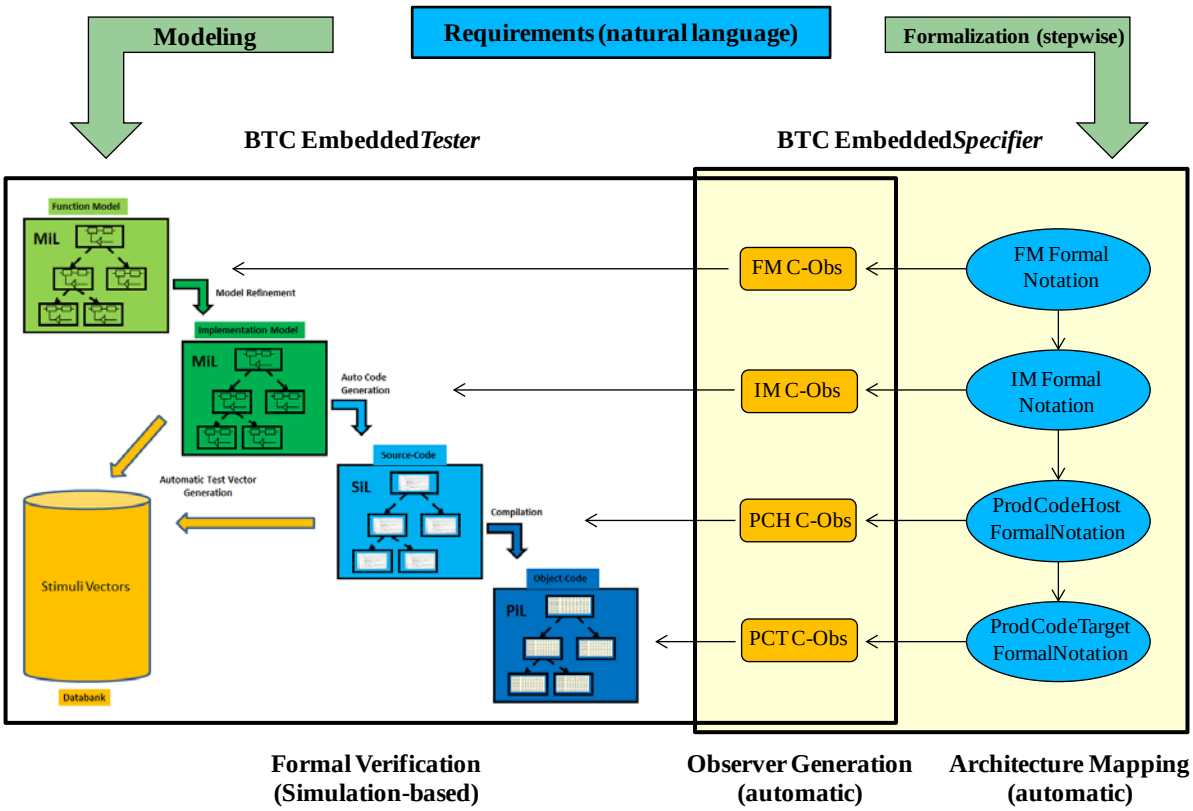

Figure 2-3: Workflow between BTC EmbeddedSpecifier and BTC EmbeddedTester

Besides these analysis capabilities, BTC EmbeddedSpecifier will provide an OSLC adapter for the RTP that allows data access for formal requirements, C-Observer and their traceability to natural language requirements.

2.6.2 BTC Embedded Tester

2.6.2.1 Tool description

BTC EmbeddedTester from BTC Embedded Systems is a test generation, execution and test management environment especially made for TargetLink® and Simulink® for automotive applications. It supports unit and module testing for Simulink®-Models, TargetLink®-Models and Target Code as part of a model based development process. Automatic test generation and code verification capabilities are supported additionally.

A test vector manager is an integral part of EmbeddedTester allowing to organize, visualize, export and debug test vectors. Groups of either stimuli vectors or/and test vectors, including reference values, can be managed. Complete test sequences can be defined to allow batch test execution.

BTC EmbeddedTester offers execution of test vectors on different levels, such as Model-in-the-loop, Software-in-the-Loop or Processor-in-the Loop in a back-to-back-testing fashion. Furthermore, vectors can be exported for further reuse, for instance HiL-Testing. It supports easy reuse of new and existing test sets from various sources. After importing existing test cases BTC EmbeddedTester shows achieved code coverage.

Test models for automatic test generation can be constructed using existing modelling tools such as Matlab Simulink/Stateflow and dSPACE TargetLink. Please refer to these tools for more information on how to model with them. BTC EmbeddedTester can also be applied directly for test vector generation on handwritten code in C.

2.6.2.2 Development plans within MBAT

The tool will be enhanced such that it can perform simulation-based formal verification both on model level and code level to verify formal specifications of requirements coming from BTC Embedded Specifier.

For this purpose, BTC EmbeddedTester will be integrated with BTC EmbeddedSpecifier (see WP2.2) to provide model based test generation together with requirement verification during test execution run time. As shown in Figure 2-3, C-Observer generated out of a formal specification are used to analyse requirement fulfilment of the SUT. This will be enabled within MBAT for the following SUT representations

- Function Model: Within MBAT this will be Matlab Simulink
- Implementation Model: Within MBAT this will be dSPACE TargetLink
- Source Code
- Target Code

This enables requirement verification from the early beginning up to the software on the target. The method will be developed to be extendable for further SUT representations (e.g. integration testing or HIL testing).

The following general workflow will be supported for all levels:

- Import C-Observer
- Import Test Vectors talking about the interface description of the current SUT representation (e.g. Simulink)
- Import SUT (e.g. Simulink)
- Simulate test vectors and record results
- Perform formal verification based on C-Observer using test vectors as behavioural description of the SUT.
- Record analysis results

As for BTC EmbeddedSpecifier, BTC EmbeddedTester will also come with an OSLC adapter to provide access to analysis and test results for test vectors and C-Observer analysis.

- Addressed use cases

Simulation based formal verification enabled together with BTC EmbeddedSpecifer and BTC EmbeddedTester addresses the following four scenarios:

- WP1.4_DAI_UC02
- WP1.4_RIC_UC01
- WP1.4_AVL_UC01
- WP1.4_AVL_UC02

2.7 CEA LIST

CEA LIST provides two tools in WP2.4: Fluctuat and its extension HybridFluctuat. Both tools are abstract interpretation based static analysers of C and ADA codes with an emphasis on numerical analysis, HybridFluctuat being in addition able to analyse hybrid systems. Within MBAT, in addition to improving the quality of the tools and the precision of the analysis by improving our abstract domains, we shall develop new functionalities that are intended to facilitate the use of both tools for model based V&V.

	M2	M3	M4
Fluctuat	Test case generation	Modular analysis Interactive analysis	
HybridFluctuat	--	Parse and analyse Simulink descriptions	Link with the Diversity tool

Table 2-7: CEA LIST tool functionalities

2.7.1 Fluctuat

2.7.1.1 Tool description

FLUCTUAT is a static analyser using abstract interpretation of ANSI C and ADA programs, that focuses on numerical properties. For that, it computes ranges of values that can be taken, for all possible executions, at all control points of the program, with two different semantics, the idealized one in real numbers, and the implemented one infinite precision numbers (here IEEE 754 floating-point numbers and machine integers). It bounds the difference between the values taken by variables with these two semantics, and decomposes it on its provenance on the control points of the program, allowing the user to determine which parts of the program mainly contribute to the rounding error.

2.7.1.2 Development plans

During MBAT, we plan to integrate the following functionalities to FLUCTUAT.

1. Test case generation.

   Once the analysis using FLUCTUAT is finished, it is possible to extract from the computed invariant a (sequence of) value(s) for the input(s) of the program that reach the maximal and/or minimal values within the invariant. These inputs may then be given to a TCG tool as special test case scenarios to be tested. This feature will help the combination of test and analysis on programs with numerical inputs. It thus will address the core requirements 909_ref_05

2. Modular Analysis

   We recently proposed a method to perform modular static analysis of large programs (see the paper at SAS 2012, [GPV, 2012]). This technique allows analysing parts of a program (for example, a function) and re-using the results of this analysis later. It thus helps to analyse large programs but also facilitates the collaboration of FLUCTUAT with other tools by computing the summary of the analysis of a function. It will address the core requirements 408_ref_01.

3. Interactive Analysis

We plan to develop a new mode in FLUCTUAT which will allow for interactively analyzing the program. The objective of this mode is to explore the parameters and the annotations in order to make the automatic mode succeed in filling the numerical requirements. The interactive mode shall present a user's interface close to the gdb's one. Hence most of the commands are inherited from gdb. The differences from gdb are the following:

- the interactive static analysis can start from any entry point;
- the analysis can explore several branches/paths in the same session;
- the result of print command are intervals with accuracy errors instead of simple values.

This feature shall be used to address the core requirements 144_ref_02 by helping the user to localize the potential defects and 372_ref_04 as it may help to export analysis results to other tools. In particular, we believe that this could lead to an interaction with the Frama-C tool.

2.7.2 HybridFluctuat

2.7.2.1 Tool description

HybridFLUCTUAT is a static analyser using abstract interpretation of hybrid systems made of a C program interacting with a physical environment described by means of ODEs. It is based on the FLUCTUAT tool and computes the same information but for more complicated systems. The communication between the discrete program and the continuous environment is modelled via annotations in the program that model the effect of sensors and actuators.

2.7.2.2 Development plans within MBAT

During MBAT, we plan to integrate the following functionalities to FLUCTUAT.

1. Use Simulink descriptions of the physical environment. For now, the dynamics of the environment must be given as a set of C++ functions that model the ODEs. We plan to integrate better HybridFLUCTUAT into Matlab/Simulink as this is the language of choice for defining dynamical systems. In particular, we plan to allow the user to define the dynamics of the physical environment with a Simulink model. This should be the first step towards a static analysis of close-loop control-command systems described in Simulink and thus reduce the cost of the detection of run-time errors. This addresses the core requirement 457_ref_01.
2. Link with the DIVERSITY tool.

   Based on the previously described functionality, we plan to develop a link between HybridFLUCTUAT and the DIVERSITY tool also developed by CEA LIST by using HybidFLUCTUAT to analyse the continuous environment and provide DIVERSITY with constraints describing it. In this way, the static analysis may help to improve the test case generation tool by taking care of the part of the model not addressed by DIVERSITY now. This feature shall address core requirements 909_ref_05, 372_ref_04 and help to increase the interoperability between tools.

2.8 Enea - ENEA

Enea is a well-known supplier of embedded real time and OSE RTOS [OSE, 2012] related solutions particularly for the telecom systems. The services provided by Enea to customers are in the form of products, training, provision of solutions and consultancy.

In the context of model-based development, a modelling and analysis solution for Non-Functional Requirements (NFRs) has been developed by the partner.

	M2	M3	M4
NFR Profile suite	Modelling concepts for NFRs, IDE, automated analysis in the form of a model transformation, preliminary OSLC compliance	Final OSLC Adapter, IDE and analysis automation improvements	Final tunings for RTP integration

Table 2-8: ENEA tool functionalities

2.8.1 NFR Profile suite

2.8.1.1 Tool description

NFR Profile suite consists of a UML profile [NFR, 2012], which provides necessary semantics for generic and high-level modelling of non-functional requirements along with important properties for them to enable analysis of trade-offs and mutual dependencies and impacts. This analysis is done on a model annotated with the NFR profile through an in-place model transformation; which basically traverses the model, performs calculations on the model based on a set of rules, and write backs the results to the same model as properties of model elements.

2.8.1.2 Development plans

Currently the profile (modelling semantics) is already finished and implemented using Papyrus in Eclipse as a prototype. A model transformation to perform sophisticated analysis on the non-functional requirements (in a generic way and regardless of the type of a non-functional requirement; i.e., availability, etc.) is also developed to demonstrate the feasibility of its analysis automation. The transformation needs to be improved and there are some parts of it which are still under development. The connection to RTP is yet to be done. Towards this goal, first a preliminary version of OSLC adaptor for the suite is planned to be developed which is then refined and finalized in M3. The IDE and analysis will also be polished by this milestone to improve its usability and user-friendliness. As the last step any potential fixes necessary to integrate and incorporate it as part of the final RTP will be performed if needed; for example, if incompatibilities are detected after M3. Moreover, application of the approach to the use case is also in-progress.

The tool helps with the combination of model-based analysis and testing by providing guidance in selection of test cases (i.e., avoiding blind selection) to focus on parts of the system which are identified as having major problems. This can help with the prioritization and optimization of testing efforts especially when there is a limited amount of resource available for testing activities. This is also aligned with the aim of MBAT to reduce time to market of the products (which implies optimizing V&V efforts taking into account limited time/budget allocated for V&V activities).

The approach can also provide some sort of impact analysis; in the sense that when a modification in the system is made such as introduction of a new requirement or a new feature, the analysis can be automatically and easily executed on the new model. Using the result of the analysis the new model of the system can be compared with the old one (before modification) and it can be observed which other parts of the system can be affected by the modification. This way, it can also be determined whether the modification is actually desirable and good considering the system as a whole or not.

Since the approach is implemented in Papyrus-Eclipse, the linking to other Papyrus-based tools is particularly straight forward. For other tools, the connection will be provided through the OSLC adapter and only if needed it can also be done through specific transformation (although such a fixed tool-specific point to point exchange mechanism is to be avoided in MBAT). Tools that are currently considered as potential points of connection to exchange information with the NFR profile suite (i.e., provider and consumer of data) could be requirement management tools in MBAT, test case repositories, and even test execution frameworks.

As for the different scenarios of the selected automotive use case (Volvo's Brake-by-wire system), the following scenarios are addressed (either partially or completely) by the approach: WP1.4_VOLVO_UC01_SC01, WP1.4_VOLVO_UC01_SC02, WP1.4_VOLVO_UC01_SC06.

2.9 Infineon Technologies Austria - IFAT

IFAT will provide the following tools.

	M2	M3	M4
DODT		RTP integration (via OFFIS)	

Table 2-9: IFAT tool functionalities

2.9.1 DODT

2.9.1.1 Tool description

DODT (Domain Ontology Design Tool) is a requirements engineering tool that was developed within the CESAR Artemis research project. It provides two main functions. The first function helps to create semantically correct requirement statements. For the specification of new requirements, a semantic guidance system makes use of domain ontology knowledge to derive textual phrases to be used by the requirements engineer. The second main function is the analysis of requirements. The domain information is compared with requirements information. The tool reports any deviations to the user and suggests improvements.

2.9.1.2 Development plans

The tool is planned to be applied on WP1.4_IFAT_UC01_SC01 for RTPv2. The integration of DODT into the MBAT RTP will be done via the OFFIS tool chain. OFFIS will support the communication with DODT via their tools. It was not planned to modify or enhance DODT within the scope of MBAT, therefore all features and functions further described in [WP2.2a, 2012] and [WP2.4a, 2012] are already available.

2.10 Mälardalen University - MDU

ViTAL, A Verification Tool for EAST-ADL Models using UPPAAL PORT (see www.vitaltool.org), provides the possibility to express functional EAST-ADL behaviour in timed automata semantics. The tool enables modelling and analysis of EAST-ADL functional models and identifies dependencies as well as potential conflicts between different system functionalities before the actual system implementation. An automatic model transformation to UPPAAL PORT tool is available, which enables model checking of EAST-ADL models in an integrated environment.

	M2	M3	M4
ViTAL	Simulation and model checking of EAST-ADL models, plus offline test-suite generation for a given coverage criteria	Simulation and model checking of EAST-ADL models, plus offline test-suite generation for various coverage criteria	Simulation and model checking of EAST-ADL models, plus suite generation and optimization

Table 2-10: MDU tool functionalities

2.10.1 ViTAL

2.10.1.1 Tool description

ViTAL is a new tool developed at Mälardalen University (MDU) within the MBAT project. It integrates the architectural description language EAST-ADL with verification techniques in order to provide model checking of EAST-ADL models with respect to timing and functional behavioural requirements.
With ViTAL, we propose an automatic model transformation to UPPAAL PORT model-checker, which enables UPPAAL PORT to handle EAST-ADL models as input language and provide functional and timing behaviour of functional blocks using timed automata semantics.

The tool's unique features are:

- Automatic transformation of system models described in EAST-ADL into an input language of the UPPAAL-PORT model-checker enriched with timed automata semantics;
- Simulation of EAST-ADL system models;
- Analysis and verification of EAST-ADL models based on both functional and extra-functional constraints;
- Counter-example visualization as an UPPAAL-PORT trace;
- Exploitation of the hierarchical structure of EAST-ADL language by using efficient model-checking analysis.

2.10.1.2 Development plans

MDU will apply the ViTAL tool on the Brake-By-Wire use case provided by VOLVO in which we will address the analysis on the analysis level scenario (WP1.4_VOLVO_UC01_SC02), providing:

- simulation,
- model checking,
- response time analysis.

In the near future we will develop ViTAL TestGen, an extension of the ViTAL tool, to provide test suite generation capabilities. The test case generation will exploit trace information resulting from

the formal analysis of EAST-ADL enriched with Timed Automata behaviour models, by prioritizing the test suite generation based on the counter-example produced by UPPAAL PORT. The test suites will be generated from UPPAAL models for different coverage criteria.

Using model-to-model transformation we will connect ViTAL to UPPAAL SMC to also provide statistical model-checking to the EAST-ADL models that have previously been enriched with timed automata semantics.

ViTAL extension with regard to MBAT goals:

- Interoperability:

  At this moment we are researching the OSLC standard and how to modify ViTAL's interface to be OSLC compliant, which will be implemented in the RTPv1 (milestone M2).

- Combination of test and analysis:

  The test case generation will exploit trace information resulted from the formal analysis of EAST-ADL enriched with Timed Automata behaviour models, by prioritizing the test suite generation based on the counter-example produced by UPPAAL PORT.

For RTPv0, MDU will apply the ViTAL tool on the Volvo Brake-By-Wire use case and provide model analysis of EAST-ADL models with respect to timing and functional behavioural requirements.

2.11 OFFIS

As an R&D institute OFFIS primary focus is on development of novel methods and technologies. Therefore, our products have mainly the status of a prototype implementation. Within the MBAT project we will extend our current technology developed in different European projects (e.g. CESAR, SPEEDS) to address the MBAT challenges.

	M2	M3	M4
Requirements engineering with RSL and the PatternEditor	Based on MBAT meta-model and IOS principles, combination of BTC and RSL pattern	Probabilistic and real-time extension of RSL pattern language	--
Requirements-based Change Impact Management	Based on IOS principles, automatic execution of entailment, integration of V&V editing functionality	Explicit contract support, extension of the approach to allocation and realization relations	Implementation of shifting approach
Model-based safety analysis	Reachability analysis on abstracted Simulink/StateFlow models	Simulation-based fault-tree generation	--
Requirements Consistency/Entailment analysis	Adaptation to Volvo use case requirements, support for BTC patterns	Dedicated IOS-based analysis service	--
Sequence diagram-based specification	Specification of test cases and requirements, consistency analysis	--	--

Table 2-11: OFFIS tool functionalities

2.11.1 Requirements engineering with RSL and the PatternEditor

2.11.1.1 General Tool description

Typically, requirements are stored as natural language text which has the disadvantage that ambiguities of the sentences can cause a huge amount of requirement changes in early as well as in later development phases. To reduce the costs coming from these ambiguities, formal languages are used to have a fixed semantic meaning for a requirement. But it does not only require some training to write requirements in these formal languages, it can also be hard to read them.

The pattern based RSL fills this gap by providing an easy to learn formal language with a fixed semantics that is still readable like natural language.

The PatternEditor for the pattern-based RSL can be used for requirements capturing or during the formalization process of requirements. The inputs for the formalization process are requirements in natural language. During the formalization process, this editor supports on-the-fly syntax checking and provides a list of available patterns. This tool communicates with DOORS or Excel and the pattern is transferred to the corresponding row. If the requirement pattern is a refinement of a natural language, a refinement-link between the informal text and the pattern is created automatically.

The editor is structured according to the contract-based approach so that assumptions and promises can be specified separately.

2.11.1.2 Development within MBAT

Within MBAT we are going to integrate this requirements engineering approach into the MBAT RTP, which includes support of the MBAT meta-model and the MBAT IOS principles. The tool will mainly be used in the Volvo use case, and application to other use cases is foreseen.

We are planning two extensions of the RSL within the MBAT project. First, we extend the RSL vocabulary (syntax) by including the pattern of the BTC EmbeddedSpecifier, which will enable interoperability between the OFFIS and BTC tool-chains. Second, we will extend the semantics of the RSL by adding support for probabilistic and real-time patterns.

2.11.2 Requirements-based Change Impact Management

2.11.2.1 General Tool description

We present an approach to an integrated change management solution for systems design that reduces the V&V activities drastically by determining the impact boundaries of a change. In contrast to existing techniques to determine change propagation our approach is not based on structural graph based data structures but on the behaviour described through the requirements and their connection to test cases. Furthermore, we propose automation techniques based on formalized requirements to guide the change management process and the selection of compensation candidates for changed system elements.

The method of dealing with changes in the system can roughly be summarized in a cyclic four-step process depicted in Figure 2-4. A change is the initial action in this cycle. To determine the impact of the change and keep the effects on other system artefacts as small as possible the local artefacts will be considered and an analysis will be performed. The a-priori identified test cases will be re-run. Based in the results compensation strategies will be suggested to get the system back in a consistent state.

The software can be used with informal requirements and manually created test cases as well as with formalized requirements and automatically created test cases. The most speed-up compared to conventional model-based design is provided using the formal requirements specification language RSL together with automated entailment analyses, that enable the tool to automatically determine change impacts across the derived requirements. Even in this case manually created test cases and verification activities like reviews and hands-on testing are supported.

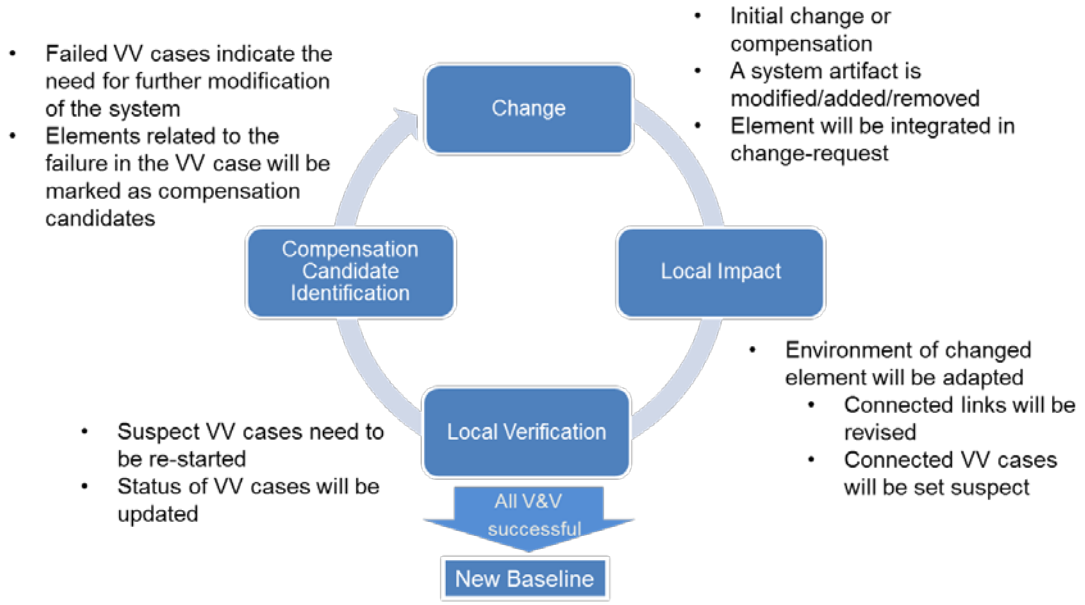

Figure 2-4: Basic process of the change impact management tool

The tool implements this approach, providing the functionalities:

- Managing of change requests
- Automatically determine affected V&V cases and reset status (mark as suspect VV cases)
- Display change requests with associated system artefacts
  - Components
  - Requirements
  - Implementations
  - VV Cases
  - Trace-links
- Re-run of VV activities
- Suggestion of compensation candidates for failed VV cases

2.11.2.2 Development within MBAT

The change management is mainly developed within the Volvo use case. The current schedule is as follows (some functionalities might still be subject to change):

M1

- The tool displays a change request and allows to manually set the V&V status
- The EAST-ADL, Simulink and Excel server are connected to the change management server.

M2

- The communication shall be performed according the IOS specified in MBAT including the meta-model and the transfer protocols.

- Automatic Entailment checking using the RSL shall be integrated. The Pattern Editor shall be used to insert requirements
- An edit functionality for V&V cases is necessary to change the type, the automation-kind and the connected system artefacts.

M3

- The contract based design principle shall be explicitly included. Requirements in form of contracts were already supported before, but the evaluation of the assumptions has not yet been used for the guidance of the change MM process.
- Furthermore the allocation and realization links in a perspective oriented design space shall be integrated. This allows the tool to be used transparently in a process across multiple structural levels of a system (use-cases, logical design, SW/HW design, and geometrical design).

M4

- We will implement shifting as a technique that allows to estimate the quality of compensation candidates. This approach is not limited to local impacts but can be spread over large parts of the system. The performance needs to be evaluated.

2.11.3 Model-based safety analysis

2.11.3.1 General Tool description

The OFFIS safety analysis platform originally developed during the projects ESACS and ISAAC [Peikenkamp, 2006]. It implements a novel approach for model based safety analysis.

The tool chain offers different safety analysis techniques such as Fault-Tree Analysis (FTA) and Failure Mode and Effect Analysis (FMEA). To ensure it can be integrated with the typical development and safety processes, analysis results are processed and presented in the same formats used by traditional tools and analysis techniques. For instance the platform provides an interface to translate FTA results into the fault tree format supported by the commercial fault tree tool FaultTree+.

A fault injection technique is used to enrich these models with the dysfunctional behaviour specifications that enable the tool chain to systematically evaluate each fault and determine its potential impact on the system.

The extended models are automatically analysed using formal verification techniques (i.e. model checking) to identify dependencies between faults, failures and hazards leading to violations of safety requirements. This approach is more comprehensive with respect to the traditional approach because it allows to consider the full temporal properties of the system. For example, the approach allows the specification of safety-critical events by using temporal patterns (RSL). These specifications can describe a specific sequence in which events have to occur in order to be safety-relevant.

In addition to the BDD-based analysis method, the tool also supports a Monte Carlo-based simulation engine. The simulation engine has been developed especially for models that are too complex for the standard model-checking engine or when it utilizes functions that are not supported by model checking. In general, the simulation engine can handle models that are several orders of magnitude more complex than the BDD engine can. The drawback with simulation is that the simulation results are always incomplete (i.e. not all cut-sets may be found and not all cut-sets are minimal).

2.11.3.2 Development within MBAT

For the Daimler use case (UC03 – adaptive brake light) we will develop BDD-based reachability analysis based on behaviour description as Simulink/StateFlow models. As a first step we will apply a formal abstraction to this model. This will reduce the complexity of the problem such that the existing BDD-based analysis methods can be applied.

To the topic of the automated fault-tree generation, we will extend the current simulation engine to guide the simulation and produce results with a given level of confidence.

2.11.4 Requirements Consistency/Entailment analysis

2.11.4.1 General Tool description

Within a component-based design approach each individual component is associated with a set of contracts it has to fulfil. These contracts specify both, the assumptions of the component to its intended environment and its promised behaviour if these assumptions are fulfilled.

A formal specification of the assumptions and promises of the contracts with the RSL pattern language enables automatic analysis procedures which support different design steps. These procedures can be used for the verification of a single component and its contracts. For instance one could check if the behaviour of a component (its implementation) fulfils all contracts which can be done by either testing or formal verification. In addition, the formal contracts can be analysed even if no implementation for a component is available. With our tool, OFFIS can address two relevant analysis scenarios: a contract consistency analysis and a contract entailment analysis.

2.11.4.2 Development within MBAT

The requirement consistency/entailment is part of the change management and the requirement engineering tools. It will be developed within the same use cases. We are going to extend the analysis to support the BTC pattern as well as the new RSL pattern.

The analysis itself will be deployable as a service within the MBAT RTP.

2.11.5 Sequence diagram-based specification

2.11.5.1 General Tool description

Sequence diagrams are a well-known formalism in the UML to model interaction between different actors in a system. There are a set of tools available, which allow the specification of sequence diagrams, but there is no prior tool available to generate test cases with a specified timing behaviour from these sequence diagrams. This technology will be developed within the project.

2.11.5.2 Development within MBAT

Within MBAT we aim to adopt the sequence diagram specification to allow the specification of test cases as a special form of interaction between a system-under-test (SUT) and test components. We are going to combine the benefits of the specification mechanism with formal semantics similar to the life sequence chart formalism. In addition, sequence diagrams can describe sets of system traces and are not limited to simple test vectors.

The specified behaviour can be either used within a V&V case in the role of a test case or as requirement describing parts of the required behaviour of the system.

The formal semantics of our sequence diagrams allows analysing if a given set of test cases/requirements for a given component is consistent.

2.12 Technische Universität München - TUM

TUM contributes to MBAT by developing techniques for ensuring non-functional safety properties of concurrent programs. These techniques are implemented in the open source Goblint analyser.

	M2	M3	M4
Goblint	OSEK Frama-C plug-in	ARINC Deadlock detection Result exchange RTP integration	Precision & Usability

Table 2-12: TUM tool functionalities

2.12.1 Goblint

2.12.1.1 Tool description

Goblint is a sound static analyser for detecting race conditions in multi-threaded code. The focus thus far has been on developing techniques that can deal with low-level operating system code. We can handle code that manipulates complicated data structures, such as arrays and linked lists.

2.12.1.2 Development plans

We currently analyse Posix Threaded C code (IEEE Std 1003.1c-1995) to detect race conditions. Within MBAT we plan the following improvements:

- OSEK: Analysing code running on an Autosar/OSEK compliant operating system. (UC A2)
- ARINC: Analysing code running on an ARINC 651 compliant operating system. (UC AE3)
- Frama-C plug-in: We plan to compile Goblint as a plug-in for the Frama-C analysis framework, developed by MBAT partners CEA. For M2, we plan only to share the same user interface and frontend; for M3, we consider deeper integration with other Frama-C analyses. (UC AE3)
- Result Exchange: For combining testing and analysis, as well as incorporating results of other analysis tools, we plan to develop semantically meaningful exchange of analysis results. This will be based on the shared models developed in task 2.2.3 of WP2.2. (UC A2)
- Deadlock detection: In addition to data races, we intend to develop methods to detect other thread safety issues. Most important for this is deadlock detection for Posix and ARINC 65. (UC AE3)
- RTP integration. Goblint will be integrated into the RTP by M3.
- Precision & Usability. Between M3 and M4, we will focus on fine-tuning the precision of the analyses and improving the overall usability of the tool for our industrial partners.

2.13 Virtual Vehicle - VIF

STSSim is a tool allowing the in-depth analysis of system behaviours by using randomly generated or predefined input sequences. The system is modelled by a set of extended symbolic transition systems (ESTS)s defining the behaviour of the system under test (SUT) and allows the usage of variables as well as time definitions.

	M2	M3	M4
STSSim	Integration into the VIF Independent Co-Simulation (ICOS) Platform	Extended UML state machine support	Implementation of static analysis functionalities (TBD)

Table 2-13: VIF tool functionalities

2.13.1 STSSim

STSSim is part of the tool suite STATION (STAte based system Test and simulatION) and can be combined with STSTest in order to generate test cases.

2.13.1.1 Tool description

STSSim can be used to model and analyse the system behaviour of timed systems. It uses symbolic techniques in order to deal with variables efficiently and relies on ESTS for component modelling. An ESTS contains extensions allowing an easy model transformation from UML state machines and to incorporate timing information. It allows the calculation of the best- and worst case execution time handles non-deterministic models and can be combined with STSTest, which is a test generation tool provided by the VIF.

2.13.1.2 Development plans

- **ICOS Integration:** STSSim will be coupled to the Independent Co-Simulation platform ICOS from ViF. ICOS allows the usage of different simulation tools in one cooperative simulation. This means that tools like MATLAB/Simulink (or Dymola) for multi-physics modelling, SystemC for Electronics, ASCET for automatic control engineering, Flowmaster for thermal simulations and STSSim for behaviour simulation are combined via the co-simulation platform to form the virtual prototype.

- **Extended UML support:** The unified modelling language (UML) has become a de-facto industry standard and supports a wide range of models. STSSim will be extended in order to support the transformation of most state machine elements defined in UML in order to foster the system modelling efficiency.

- **Implementation of static analysis functionalities:** STSSim will be extended by static analysis functions in order to enhance the correctness of the modelled ESTSs and to provide control mechanisms for test case generation.

- **RTP integration:** STSSim will be relying on OSLC and related technologies in order to communicate and to be compatible with other tools deployed in the MBAT RTP.

3 Abbreviations and Definitions

ADL	Architecture Description Language
AIS	AbsInt Annotation Language for aiT/StackAnalyzer
ARINC	(System Architecture) Specifications for Integrated Modular Avionics
BBW	Brake By Wire
BDD	Binary Decision Diagram
BTC-ET	BTC Embedded Tester
DTF	Data Flow Time (Simulator)
ECU	Electronic Control Unit
FTA	Fault Tree Analysis
GDB	GNU Debugger tool
IDE	Integrated Development Environment
IOS	Interoperability Specification
LTL	Linear Temporal Logic
LTS	Labelled Transition Systems
MARTE	Modeling and Analysis of Real-Time and Embedded Systems – UML profile
MBAT	Combined Model-based Analysis and Testing of Embedded Systems
MiL	Model in the Loop
NFR	Non-Functional Requirement
ODE	Ordinary Differential Equation
OSE	Operating System Embedded
OSEK	Open Systems and their Interfaces for the Electronics in Motor Vehicles
OSLC	Open Services for Lifecycle Collaboration
PiL	Processor in the Loop
RSL	Requirement Specification Language
RTOS	Real-Time Operating System
RTP	Reference Technology Platform
SiL	Software in the Loop
SMC	Statistical Model Checking
SUT	System Under Test
TCTL	Timed Computation Tree Logic
TCG	Test Case Generation
UC	Use Case
WCET	Worst-Case Execution Time
WEFACT	Workflow Engine for Analysis, Certification and Test

4 References

For further information and references about individual tool components and techniques we refer to D_WP2.4_3_1 “Prototype Tools for Model-based Analysis”.

[GPV, 2012] Eric Goubault, Sylvie Putot and Franck Vedrine; *Modular Static Analysis with Zonotopes*; Proceedings of SAS 2012.

[NFR, 2012] Mehrdad Saadatmand, Antonio Cicchetti, Mikael Sjödin; *Toward Model-Based Trade-off Analysis of Non-Functional Requirements*; 38th Euromicro Conference on Software Engineering and Advanced Applications(SEAA), Cesme-Izmir, Turkey, September, 2012

[OSE, 2012] ENEA OSE, http://www.enea.com/software/products/rtos/ose/, Accessed: Nov 2012

[Peikenkamp, 2006] Thomas Peikenkamp, Antonella Cavallo, Laura Valacca, Eckard Bde, Matthias Pretzer & E. Moritz Hahn; *Towards a Unified Model-Based Safety Assessment*. Proceedings of SAFECOMP’06, pp. 275–288.

[WP1.3a, 2012] MBAT Consortium; *Measurement plans*; D_WP1.3_1_1

[WP1.4a, 2012] MBAT Consortium; *Automotive Use cases & demonstrators specification*; D_WP1.4_1

[WP1.5a, 2012] MBAT Consortium; *Aerospace Use cases & demonstrators specification;* D_WP1.5_1

[WP1.6a, 2012] MBAT Consortium; *Rail Use cases and demonstrators specification;* D_WP1.6_1

[WP2.1a, 2012] MBAT Consortium; *Refined Requirements*; D_WP2.1_1_1

[WP2.2a, 2012] MBAT Consortium; *Report and tool prototypes on MBAT models and their construction approaches needed for the Fast Track RTP*; D_WP2.2_1_1

[WP2.3a, 2012] MBAT Consortium; *Specification of Model-based Test Case Generation and Execution Methods and Tools;* D_WP2.3_2

[WP2.3b, 2012] MBAT Consortium; *Prototype tools for Model-based Test Case Generation and Execution*; D_WP2.3_3_1

[WP2.4a, 2012] Requirements for use of WP2.4 results in the RTP; D_WP2.4_1_1

[WP2.4b, 2012] MBAT Consortium; *Prototype tools for model-based analysis*; D_WP2.4_3_1

[WP3.2a, 2012] MBAT Consortium; *Specifications for RTP Interoperability*; D_WP3.2_2_1